An Adaptation of Soucy and Mineau weighting for Large Scale Text Classification

Cristóbal Camarero Coterillo

Universidad de Cantabria

Abstract. I have implemented a kNN for the LSHTC, based on the weighting for text categorization of Soucy [1] adding distance-weighting and some acceleration tricks.

1 Introduction

In this paper I simply got a kNN approach, using the weighting of components of Soucy and Mineau instead of the tipical tf-idf and did the necessary modifications to work with tens of thousands categories, and a space with dimension of hundreds of thousands.

I've made my implementation in python, which make the development easier but have a great impact on performance, forcing me to search ways to increase speed. I too show the tuning of some constants.

2 Method

2.1 ConfWeight[1] summary

Here I will describe shortly the weighting of Soucy and Mineau I have used.

The proportion of documents containing a feat can be estimated as:

$$\tilde{p} = \frac{x + \tau}{n + 2\tau}$$

Being $\tau = 0.5 z^2_{\alpha/2}$, α the confidence and $\Phi(z_{\alpha/2}) = \alpha/2$ (student's law). They recommend an $\alpha = 0.95$, which is a $\tau = 1.96$. I will tune this value for our problem.

For each label and feat, $\tilde{p}_+$ will be $\tilde{p}$ with x being the number of vectors with that label and feat and n the number of vectors with that label. And $\tilde{p}_-$ is $\tilde{p}$ with x the number of vectors with that feat but without that label and n the number of vectors without that label.

The confidence interval has the form

$$\tilde{p} \pm \tau \sqrt{\frac{\tilde{p}(1-\tilde{p})}{n + 2\tau}}$$

I will call $\overline{\tilde{p}}$ to the upper limit and $\underline{\tilde{p}}$ to the lower one.

Now the *strength* of a feat for a label is defined as

$$str_{f,l} = \begin{cases} log_2(2\frac{\underline{\tilde{p}_+}}{\underline{\tilde{p}_+} + \overline{\tilde{p}_-}}) \text{ if } \underline{\tilde{p}_+} > \overline{\tilde{p}_-} \\ 0 \text{ otherwise} \end{cases}$$

And the *stength* del feat en general es el máximo

$$str_f = \left(\max_l str_{f,l}\right)^2$$

The ConfWeight of a feat f in a vector v is

$$ConfWeight_{f,v} = \log(tf_{f,v} + 1)str_f$$

The ConfWeight has the role which usually has the tf-idf: to change the original space of the term frequencies to one in which the components are more similar among themselves.

2.2 Distance Weighting

The distance I've used is the Cosine similarity

$$\frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}$$

This has a value of 1 when the vectors are equal and 0 when are completely different, so is not properly a distance, to have a distance a make $d(x, y) = 1 - cossim(x, y)$.

The training vectors closest to the one classifying usually have all different labels, being the best to choose the closest. But when there are labels with more than one vector we need a weighting. I test with the weightings $1/(1 + d)$, $1 - d$, $1/d$, $(1 - d)^2$, $(1 - d)^3$, $(1 - d)^4$ and e^{1-d}. The best results in the dry-run dataset are obtained with the $(1 - d)^3$, but in the larger is with $(1 - d)^2$.

2.3 Speeding

With the huge number of dimensions the usual algorithms for finding the closest vectors (like Quadtree or kd-tree) don't work.

Instead, I have used a very simple thing, test for proximity only the vectors that have not-null at least one of the ν greatest components of the vector classifying. Unexpectly, I got the optimal in $\nu = 3$ instead of with $\nu = \infty$.

3 Results

Here I show some of the results on the dry-run data.

Importance of the confidence:

τ	accuracy
0.0	0.449946
1.0	0.453714
1.5	0.456405
1.96	0.462863
2.25	0.463402
2.5	0.470936
2.625	0.472013
2.75	0.467707
3.0	0.462325
5.0	0.313778

So, in this case the best is around $\tau = 2.625$ instead of $\tau = 1.96$ as mentioned by Soucy. This correspond with a confidence interval at 99.1%.

The number of neighbours considered

k	accuracy
1	0.398816
3	0.409580
4	0.409580
5	0.406351
6	0.411195
7	0.409580
10	0.404198

Only using vectors with not-null at least one of the ν greatest component of the classifying.

ν	accuracy
2	0.358450
3	0.367061
4	0.363832
5	0.365447

Comparison of the distances

distance	accuracy
$(1-d)$	0.461787
$(1-d)^2$	0.472013
$(1-d)^3$	0.475242
$(1-d)^4$	0.470936
e^{1-d}	0.458019

But in the larger dataset the $(1-d)^2$ distance is a little better than the $(1-d)^3$.

And here I show the respective accuracy of the tf-idf and ConfWeight weightings, each one in its best conditions (except both with distance weighting of $1-d$).

weighting	accuracy
tf-idf	0.311087
ConfWeight	0.461787

We see a great improvement only because changing the weighting.

4 Time Measures and Complexity

My equipment:

Processor	AMD Turion(tm) 64 Mobile ML
OS	Windows XP 32 bits
Memory	1GB
frequency	1600 MHz

Both the training and testing run in less than a minute for the dry-run dataset. On the larger data the training last about 15minutes and the testing about 2 hours.

Being S the total trainfiles size, F the number of feats and L the number of labels (classes) the complexity of the training is $O(S+FL)$.

For training it is more complex, for each one of the test vectors (n) you need to transform by the ConfWeight ($O(f)$, having a mean of f feats), get the ν greater components ($O(f\log(\nu))$) and collect the corresponding vectors (v, which we only can bound by the train vectors, but it is a lot smaller), then I calculate the distances to each ($O(vf)$) and get the k nearest neighbours ($O(v\log(k))$), I weight each label by the distance and get the greatest ($O(k)$). So, in total is $O(n(f\log\nu + vf + v^2\log v + k))$

5 Conclusions

I show that the ConfWeight is a great improvement over the tf-idf, and the necessity of a distance weighting.

python is a good language to develop the algorithms, but to have good performance it should be rewritten in C/C++.

References

1. P. Soucy, G. W. Mineau. “Beyond TFIDF Weighting for Text Categorization in the Vector Space Model” In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005).